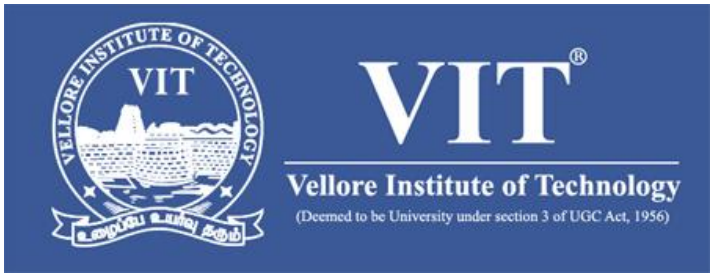


(Estd. u/s 3 of UGC Act 1956) Vellore - 632 014, Tamil Nadu, India
School of Computer Science and Engineering

PYTHON FLASK APP WITH DOCKER

Dhruv Mittal 17BCE2110
Supragya Raturi 17BCE0866
Under guidance of Prof. Siva Shanmugam
Virtualization, Vellore Institute of Technology, Vellore P.O.BOX-63201

ABSTRACT

In today's world, where many applications are deployed on the various clouds, our project aims to deploy the famous python flask application API on a container image with the help of Docker. Basically, it is a Python based web framework for increasing the ease in making various web portals and creating URL routes. As our application becomes bigger and more complex, we need to simplify the deployment process through automation. Creating a whole mega LINUX instance for specific pre-defined dependencies-based application is not affordable and liable. We need to have to only the required and sufficient software's and hardware access.

INTRODUCTION

Docker is an open-source application that allows administrators to create, manage, deploy, and replicate applications using containers. Containers can be thought of as a package that houses dependencies that an application requires to run at an operating system level. This means that each application deployed using Docker lives in an environment of its own and its requirements are handled separately.

Flask is a web micro-framework that is built on Python. It is called a micro-framework because it does not require specific tools or plug-ins to run. The Flask framework is lightweight and flexible, yet highly structured, making it preferred over other frameworks.

Deploying a Flask application with Docker will allow you to replicate the application across different servers with minimal reconfiguration.In this tutorial, you will create a Flask application and deploy it with Docker.

VIRTUALIZATION

Virtualization is the division of physical computer and networking resources into smaller, more flexible units, presenting these smaller units to users as though each was a discrete resource. The idea is that , instead of assigning specific computing tasks to individual physical servers which may sometimes end up being over or under used – a single physical server can be logically divided into as few or as many virtual servers as needed.That means, as the figure below illustrates, there can be dozens of individually installed operating systems (OS) running side by side on the same hard drive. Each OS is effectively unaware that it isn't all alone in its local environment.

Practically, each operating system instance can be accessed remotely by both administrators and customers in exactly the same way as any other server.In this kind of environment, as soon as your virtual server completes its task or becomes unnecessary, you can instantly delete it. This will free up the resources it was using for the next task in the queue.There's no need to over-provision virtual servers to anticipate possible future needs, because future needs can be easily met whenever they arrive.In fact, today's virtual server might only live a few minutes or even seconds before, having completed its task, being shut down for good to make room for whatever's next. All this allows for far more efficient use of expensive hardware. It provides the ability to provision and launch new servers at will, either to test new configurations or add fresh power to the production services.

Cloud computing providers like AWS use virtualized computers of one kind or another. The hundreds of thousands of Amazon EC2 instances, for example, all run on top of the open source Xen or KVM hypervisors—which are themselves installed and running on many thousands of physical servers maintained in Amazon's vast server farms.Whatever hypervisor technology is being used, the goal is to provide a largely automated hosting environment for multiple complete, self-contained virtual computers.Containers like Docker, on the other hand, aren't standalone virtual machines but are modified file systems sharing the operating system kernel of their physical host.

CONTAINERS:

Containers are extremely lightweight virtual servers that, as you can see from the figure, rather than running as full operating systems, share the underlying kernel of their host OS. Containers can be built from plain-text scripts, created and launched in seconds, and easily and reliably shared across networks. Container technologies include the Linux Container project, which was Docker's original inspiration.

The script-friendly container design makes it easy to automate and remotely manage complex clusters of containers, often deployed as microservices.

DOCKERS:

A Docker container is an image whose behavior is defined by a script. The container is launched as a software process that's cunningly disguised as a server. An image is a software file containing a snapshot of a full operating system file system. Everything necessary to launch a viable virtual server is included. An image might consist of just a base operating system like Ubuntu Linux, or the tiny and super-fast Alpine Linux. But an image could also include additional layers with software applications like web servers and databases. No matter how many layers an image has and how complicated the relationships between them might be, the image itself never changes.

When, as shown in the next figure, an image is launched as a container, an extra writable layer is automatically added into which the record of any ongoing system activity is saved.

Usually Docker containers are loaded up with some kind of app development project to test how it will work, and then share it with team members for feedback and updates. When the app is complete, it can be launched as a cluster of containers (or "swarm" as Docker calls it) that can be programmatically and instantly scaled up or down according to user demand.

While Docker is a Linux-based technology and requires a Linux kernel to run, running remote or even local Docker containers on Mac or Windows machines is possible through either the Docker for Mac or Docker for Windows apps or for older machines, through the Docker Machine tool.

PROBLEM DESCRIPTION:

Deploying a multi-dependencies Python Application API on a container image with the help of Docker.

SOFTWARE REQUIREMENTS:

• Ubuntu 16.04

• CS Docker Engine 1.13, or EE Daemon 17.03 and higher.

• 8.00 GB of RAM for manager nodes or nodes running DTR.

• 4.00 GB of RAM for worker nodes.

HARDWARE REQUIREMENTS:

• 4GB free RAM for Docker

• 16GB free disk space

• CPU 4×core, 2.0GHz

LITERATURE SURVEY:

• Docker provide some facilities, which are useful for developers and administrators. It is an open platform can be used for building, distributing, and running applications in a portable, lightweight runtime and packaging tool, known as Docker Engine. It also provide Docker Hub, which is a cloud service for sharing applications. Costs can be reduced by replacing traditional virtual machine with docker container. It excellently reduces the cost of rebuilding the cloud development platform.

Docker automates the applications once they area unit pack. an additional layer of stevedore engine is additional to the host software package. The performance of stevedore is quicker than virtual machines because it has no guest software package and fewer resource overhead.

• Cloud computing is an increasingly popular paradigm for accessing computing resources. In practice, cloud service providers tend to offer services that can be grouped into three categories: software as a service, platform as a service, and infrastructure as a service. This paper discuss the characteristics and benefits of cloud computing. It proceeds to discuss the Infrastructure as a service (IaaS). This paper aims to provide a means of understanding and investigating IaaS. This paper also outlines the responsibilities of IaaS provider and the facilities to IaaS consumer. Amazon Elastic Cloud is one of the flexible clouds discussed in paper.

• Containerization is virtualization at operating system level as opposed to full machine virtualization. Docker is a tool that is used to create, ship and run applications inside containers. A single host can have multiple containers running on it. The containers may have to communicate with each other and by using docker networking, this can be accomplished. In this paper, docker networking has been explored and analyzed how it is based on the concept of Linux network namespaces. Cisco Packet Tracer is used to implement a sample network. Then the same network will be implemented in Linux using ip tool. The same network will then be implemented using docker and the relationship between network namespaces and docker networking will be established.

• Containers enable a standard, fast, and easy way of describing experiments and environments .Containers help your future self, reviewers, and other researchers and colleagues to make use and improve your work. In this paper the authors want to give an insight into the conceptual, more abstract notion of reproductibility and the concrete instructions to aid reproductibility in research and industry.

• Docker's prospect of lightweight packaging and deploying of applications and dependencies is an exciting one, and it is quickly being adopted by the Linux community and is making its way into production environments. For example, Red Hat announced in December support for Docker in the upcoming Red Hat Enterprise Linux 7. However, Docker is still a young project and is growing at breakneck speed. It is going to be exciting to watch as the project approaches its 1.0 release, which is supposed to be the first version officially sanctioned for production environments. Docker relies on established technologies, some of which have been around for more than a decade, but that doesn't make it any less revolutionary.

• In a survey, conducted by O'Reilly Media in collaboration with Ruxit throughout May 2015, invited people from the O'Reilly community to share why and how their organizations currently use—or plan to use—containers. The respondents come from a range of industries, including software, consulting, publishing and media, education, cloud services, hardware, retail, and government, among others.The results reveal that container technologies—and Docker in particular—are rapidly being adopted to make application deployment faster, easier, and more flexible. If there is a consideration adopting containers, this report will help understand why many organizations are moving in that direction and how they're doing it, along with the obstacles and challenges they must overcome.

• Cloud computing is gaining more and more traction as a deployment and provisioning model for software. While a large body of research already covers how to optimally operate a cloud system, there will still lack insights into how professional software engineers actually use clouds, and how the cloud impacts development practices. This paper reports on the first systematic study on how software developers build applications for the cloud. We conducted a mixed-method study, consisting of qualitative interviews of 25 professional developers and a quantitative survey with 294 responses. The results show that adopting the cloud has a profound impact throughout the software development process, as well as on how developers utilize tools and data in their daily work. Among other things, it was found that (1) developers need better means to anticipate runtime problems and rigorously define metrics for improved fault localization and (2) the cloud offers an abundance of operational data, however, developers still often rely on their experience and intuition rather than utilizing metrics. From these findings, a set of guidelines have been extracted for cloud development and identified challenges for researchers and tool vendors.

IMPLEMENTATION

Project api.py

```
from flask_restful import Resource
import logging as logger

class ProjectAPI(Resource):

    def get(self):

        logger.debug("Inside the post method of Task")

        projectDetails = {
                "version" : "1.0.0.0",
                "owner" : "get",
                "projectName" : "Python Flask Docker Container"
        }


        return projectDetails,200

    def post(self):

        logger.debug("Inside the post method of Task")

        projectDetails = {
                "version" : "1.0.0.0",
                "owner" : "post",
                "projectName" : "Python Flask Docker Container"
        }


        return projectDetails,200
```

init.py

```
from flask_restful import Api
from app import flaskAppInstance
from .ProjectAPI import ProjectAPI
Screenshot from 2019-11-08 21-54-44


restServerInstance = Api(flaskAppInstance)

restServerInstance.add_resource(ProjectAPI,"/")
```

app.py

```
from flask import Flask
import logging as logger
logger.basicConfig(level="DEBUG")


flaskAppInstance = Flask(__name__)



if __name__ == '__main__':

    logger.debug("Starting Flask Server")
    from api import *
    flaskAppInstance.run(host="0.0.0.0",port=5000,debug=True,use_reloader=True)
```

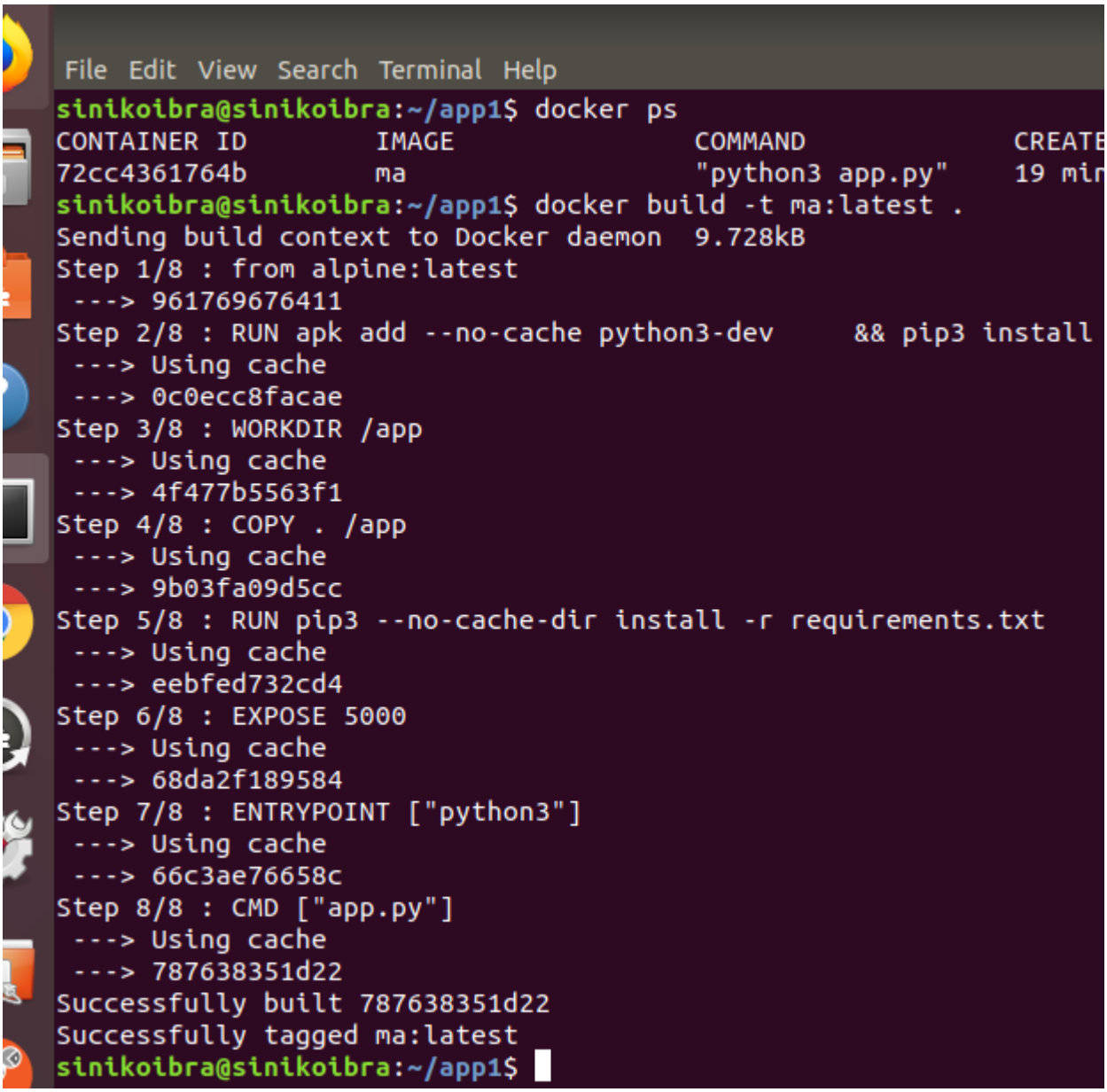
Dockerfile

```
from alpine:latest

RUN apk add --no-cache python3-dev \
    && pip3 install --upgrade pip

WORKDIR /app

COPY . /app

RUN pip3 --no-cache-dir install -r requirements.txt

EXPOSE 5000

ENTRYPOINT ["python3"]
CMD ["app.py"]
```

Rquirement's.txt

```
aniso8601==3.0.2
click==6.7
Flask==1.0.2
Flask-RESTful==0.3.6
itsdangerous==0.24
Jinja2==2.10
MarkupSafe==1.0
pytz==2018.5
six==1.11.0
Werkzeug==0.14.1
```

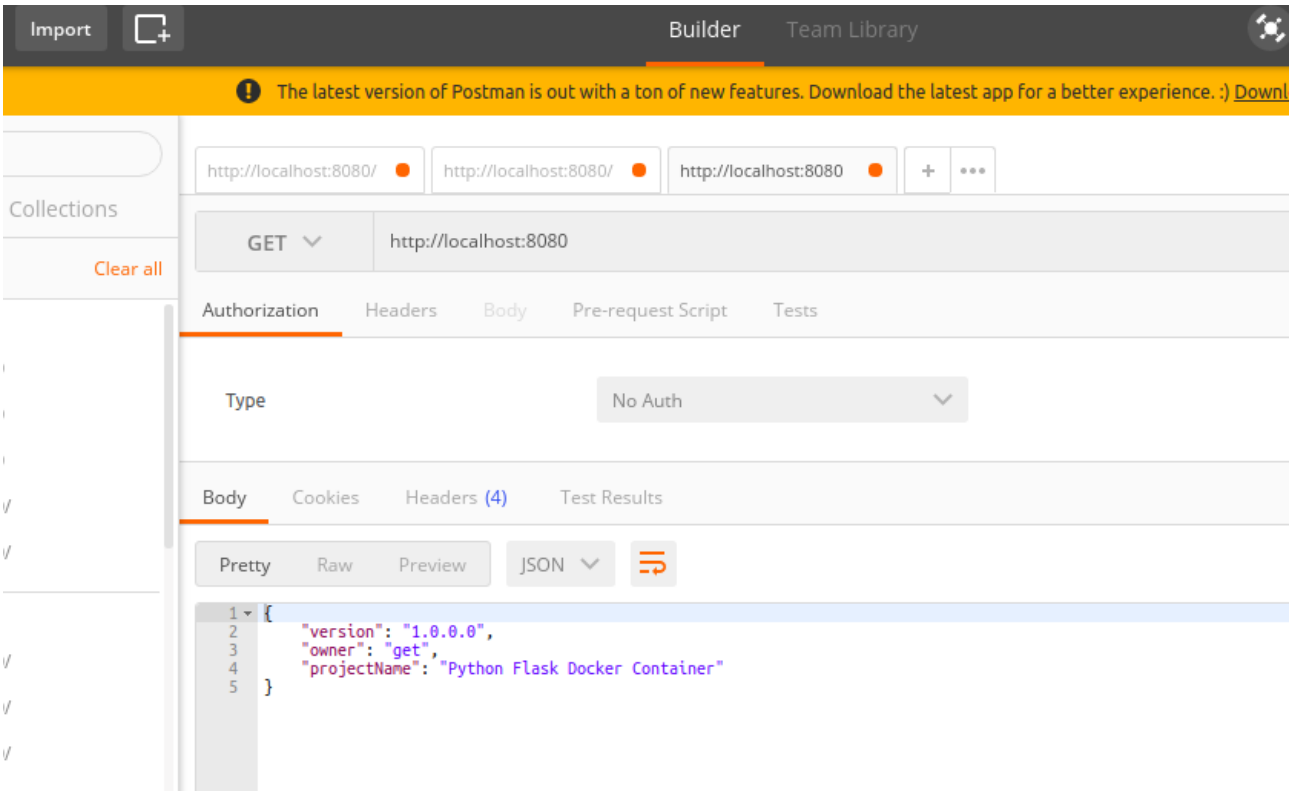
OUTPUT

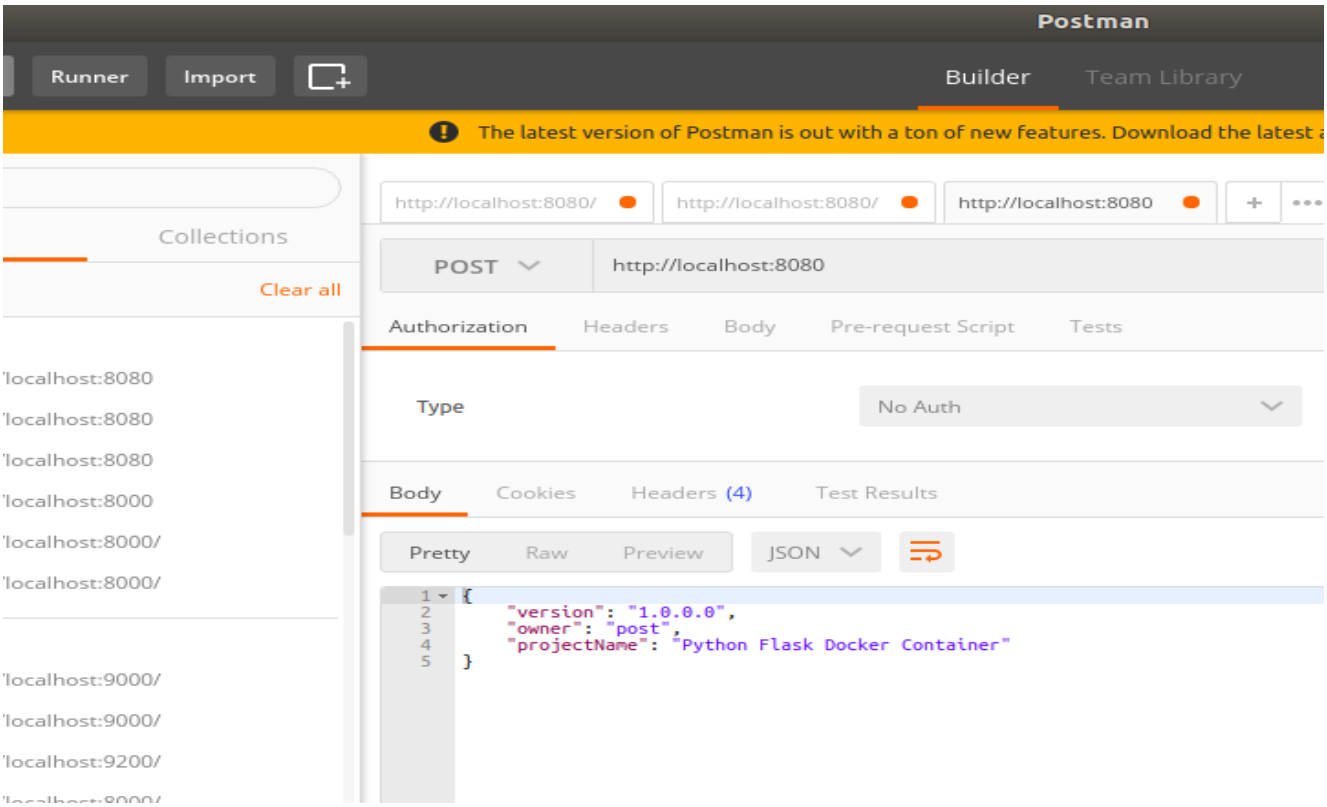

File Edit View Search Terminal Help
sinikoibra@sinikoibra:~/app1$ docker ps
CONTAINER ID IMAGE COMMAND CREATE
72cc4361764b ma "python3 app.py" 19 min
sinikoibra@sinikoibra:~/app1$ docker build -t ma:latest .
Sending build context to Docker daemon 9.728kB
Step 1/8 : from alpine:latest
---> 961769676411
Step 2/8 : RUN apk add --no-cache python3-dev && pip3 install
---> Using cache
---> 0c0ecc8facae
Step 3/8 : WORKDIR /app
---> Using cache
---> 4f477b5563f1
Step 4/8 : COPY . /app
---> Using cache
---> 9b03fa09d5cc
Step 5/8 : RUN pip3 --no-cache-dir install -r requirements.txt
---> Using cache
---> eebfed732cd4
Step 6/8 : EXPOSE 5000
---> Using cache
---> 68da2f189584
Step 7/8 : ENTRYPOINT ["python3"]
---> Using cache
---> 66c3ae76658c
Step 8/8 : CMD ["app.py"]
---> Using cache
---> 787638351d22
Successfully built 787638351d22
Successfully tagged ma:latest
sinikoibra@sinikoibra:~/app1$

```
  import      Import the contents from a tarball to create a filesystem image
  info        Display system-wide information
  inspect     Return low-level information on Docker objects
  kill        Kill one or more running containers
  load        Load an image from a tar archive or STDIN
  login       Log in to a Docker registry
  logout      Log out from a Docker registry
  logs        Fetch the logs of a container
  pause       Pause all processes within one or more containers
  port        List port mappings or a specific mapping for the container
  ps          List containers
  pull        Pull an image or a repository from a registry
  push        Push an image or a repository to a registry
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  rmi         Remove one or more images
  run         Run a command in a new container
  save        Save one or more images to a tar archive (streamed to STDOUT by default)
  search      Search the Docker Hub for images
  start       Start one or more stopped containers
  stats       Display a live stream of container(s) resource usage statistics
  stop        Stop one or more running containers
  tag         Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
  top         Display the running processes of a container
  unpause     Unpause all processes within one or more containers
  update      Update configuration of one or more containers
  version     Show the Docker version information
  wait        Block until one or more containers stop, then print their exit codes

Run 'docker COMMAND --help' for more information on a command.
sinikoibra@sinikoibra:~$ docker run -it -d -p 8080:5000 ma
72cc4361764bac1df2d0e3ee2f0385d085c1ad8244bf5dc9f077fe47c608e06b
sinikoibra@sinikoibra:~$
```

```
sinikoibra@sinikoibra:~$ docker images
REPOSITORY                          TAG                 IMAGE ID            CREATED             SIZE
ma                                  latest              787638351d22        2 weeks ago         112MB
secondtry                           latest              55594b3d020d        2 weeks ago         112MB
lvthillo/python-flask-docker-master   latest              1ff47f2cb533        2 weeks ago         923MB
flaskapp                            latest              ed6721970c98        2 weeks ago         130MB
python                              3.6                 a2e9f0fba405        2 weeks ago         913MB
ubuntu                              latest              cf0f3ca922e0        2 weeks ago         64.2MB
alpine                              latest              961769676411        2 months ago        5.58MB
lvthillo/python-flask-docker        latest              df738ffb2ef7        14 months ago       928MB
sinikoibra@sinikoibra:~$
```

```
sinikoibra@sinikoibra:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS                    NAMES
72cc4361764b        ma                  "python3 app.py"    9 minutes ago       Up 9 minutes        0.0.0.0:8080->5000/tcp   quirky_panini
sinikoibra@sinikoibra:~$
```


Import
Builder
Team Library
The latest version of Postman is out with a ton of new features. Download the latest app for a better experience. :) Downl
Collections
Clear all
http://localhost:8080/
http://localhost:8080/
http://localhost:8080
GET
http://localhost:8080
Authorization
Headers
Body
Pre-request Script
Tests
Type
No Auth
Body
Cookies
Headers (4)
Test Results
Pretty
Raw
Preview
JSON
{
"version": "1.0.0.0",
"owner": "get",
"projectName": "Python Flask Docker Container"
}


Postman
Runner
Import
Builder
Team Library
The latest version of Postman is out with a ton of new features. Download the latest
Collections
Clear all
localhost:8080
localhost:8080
localhost:8080
localhost:8000
localhost:8000/
localhost:8000/
localhost:9000/
localhost:9000/
localhost:9200/
http://localhost:8080/
http://localhost:8080/
http://localhost:8080
POST
http://localhost:8080
Authorization
Headers
Body
Pre-request Script
Tests
Type
No Auth
Body
Cookies
Headers (4)
Test Results
Pretty
Raw
Preview
JSON
{
"version": "1.0.0.0",
"owner": "post",
"projectName": "Python Flask Docker Container"
}

CONCLUSION AND FUTURE DIRECTIONS:

Docker is very convenient and Time Saving if used properly. Some applications that can only be run on a particular OS can be run easily if we have docker installed which would eventually save lot of time and memory space.

Through docker we saw implementation of single image implementation and multiple image implementation so as to run as many as application the user wants to learn without going through the hassle of installing different operating system to run a single file.

Through docker image we can run application of different domains like Machine Learning, Artificial Intelligence and Python on single docker only, which would eventually create docker images (Single and multiple) as per the requirement using basic docker commands like docker build, docker run.

REFERENCES:

[1] Babak Bashari Rad, Harrison John Bhatti, Mohammad Ahmadi, "An Introduction to Docker and Analysis of its Performance", University of Technology and Innovation Technology Park Malaysia, Kuala Lumpur, Mala IJCSNS International Journal of Computer Science and Network Security, VOL.17 No.3, March 2017s .

[2] gurudatt kulkarni, ramesh sutar.jayant gambhir, "Cloud computing infrastructure as serviceamazon ec2", international journal of engineering research and applications (ijera) , issn: 2248-9622, www.ijera.com, vol. 2, issue 1, janfeb 2012, pp.117-125 .

[3] Samyukta S Hegde, Dr. P Jayarekha, "Basic Analysis of Docker Networking", International Journal of Advanced Research in Computer Science, ISSN No. 0976-5697, Volume 8, No. 5, May – June 2017.

[4] Vincenzo Ferme, Harald C. Gall, "Using Docker Containers to Improve Reproducibility in Software and Web Engineering".

[5] Dirk Merkel "Docker: Lightweight Linux Containers for Consistent Development and Deployment",May 19, 2014.

[6] Anna Gerber," The State of Containers and the Docker Ecosystem: 2015", November 2015.

[7] Jürgen Cito, Philipp Leitner, Thomas Fritz, Harald C. Gall, "The making of cloud applications: an empirical study on software development for the cloud", University of Zurich, Switzerland .